

APPLICATION
FOR
UNITED STATES LETTERS PATENT

TITLE: NETWORK STREAMING OF MULTI-APPLICATION PROGRAM

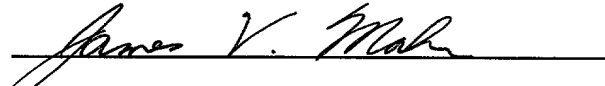
CODE

APPLICANTS: ITZIK ARTZI,
BRIDGET McDERMOTT,
DAN EYLON,
AMIT RAMON,
YEHUDA VOLK.

"EXPRESS MAIL" Mailing Label Number EL478577344US

Date of Deposit December 30, 2000

I hereby certify under 37 CFR 1.10 that this correspondence is being deposited with the United States Postal Service as "**Express Mail Post Office to Addressee**" with sufficient postage on the date indicated above and is addressed to the Assistant Commissioner for Patents, Washington, D.C. 20231.



JAMES V. MAHON, REG. NO. 41,966

Attorney Docket 006032/00024

NETWORK STREAMING OF MULTI-APPLICATION PROGRAM CODE

CROSS-REFERENCE(S) TO RELATED APPLICATIONS

The present application is a continuation-in-part of U.S. Patent Application entitled
5 “Preprocessed Applications Suitable For Network Streaming Applications and Method For
Producing Same” Filed on December 28, 2000, and assigned Serial Number <<Unknown>>.

FIELD OF THE INVENTION:

The present invention is directed to a method and system for preprocessing and
packaging application files for use by a network-based streaming application service
10 provider.

BACKGROUND OF THE INVENTION

The Internet, and particularly the world-wide-web, is a rapidly growing network of
interconnected computers from which users can access a wide variety of information. Initial
widespread use of the Internet was limited to the delivery of static information. A newly
15 developing area of functionality is the delivery and execution of complex software
applications via the Internet. There are two basic techniques for software delivery, remote
execution and local delivery, e.g., by downloading.

In a remote execution embodiment, a user accesses software which is loaded and
executed on a remote server under the control of the user. One simple example is the use of
20 Internet-accessible CGI programs which are executed by Internet servers based on data
entered by a client. A more complex systems is the Win-to-Net system provided by Menta

Software. This system delivers client software to the user which is used to create a Microsoft Windows style application window on the client machine. The client software interacts with an application program executing on the server and displays a window which corresponds to one which would be shown if the application were installed locally. The client software is
5 further configured to direct certain I/O operations, such as printing a file, to the client's system, to replicate the "feel" of a locally running application. Other remote-access systems, such as provided by Citrix Systems, are accessed through a conventional Internet Browser and present the user with a "remote desktop" generated by a host computer which is used to execute the software.

10 Because the applications are already installed on the server system, remote execution permits the user to access the programs without transferring a large amount of data.

However, this type of implementation requires the supported software to be installed on the server. Thus, the server must utilize an operating system which is suitable for the hosted software. In addition, the server must support separately executing program threads for each
15 user of the hosted software. For complex software packages, the necessary resources can be significant, limiting both the number of concurrent users of the software and the number of separate applications which can be provided.

In a local delivery embodiment, the desired application is packaged and downloaded to the user's computer. Preferably, the applications are delivered and installed as appropriate
20 using automated processes. After installation, the application is executed. Various techniques have been employed to improve the delivery of software, particularly in the automated selection of the proper software components to install and initiation of automatic software downloads. In one technique, an application program is broken into parts at natural

division points, such as individual data and library files, class definitions, etc., and each component is specially tagged by the program developer to identify the various program components, specify which components are dependent upon each other, and define the various component sets which are needed for different versions of the application.

5 One such tagging format is defined in the Open Software Description ("OSD") specification, jointly submitted to the World Wide Web Consortium by Marimba Incorporated and Microsoft Corporation on August 13, 1999. Defined OSD information can be used by various "push" applications or other software distribution environments, such as Marimba's Castanet product, to automatically trigger downloads of software and ensure that
10 only the needed software components are downloaded in accordance with data describing which software elements a particular version of an application depends on.

 Although on-demand local delivery and execution of software using OSD / push techniques is feasible for small programs, such as simple Java applets, for large applications, the download time can be prohibitively long. Thus, while suitable for software maintenance,
15 this system is impractical for providing local application services on-demand because of the potentially long time between when the download begins and the software begins local execution.

 Recently, attempts have been made to use streaming technology to deliver software to permit an application to begin executing before it has been completely downloaded.
20 Streaming technology was initially developed to deliver audio and video information in a manner which allowed the information to be output without waiting for the complete data file to download. For example, a full-motion video can be sent from a server to a client as a linear stream of frames instead of a complete video file. As each frame arrives at the client,

it can be displayed to create a real-time full-motion video display. However, unlike the linear sequences of data presented in audio and video, the components of a software application may be executed in sequences which vary according to user input and other factors.

5 To address this issue, as well as other deficiencies in prior data streaming and local software delivery systems, an improved technique of delivering applications to a client for local execution has been developed. This technique is described in co-pending U.S. Patent Application Serial No. 09/120,575, entitled "Streaming Modules" and filed on July 22, 1998. In a particular embodiment of the "Streaming Modules" system, a computer application is
10 divided into a set of modules, such as the various Java classes and data sets which comprise a Java applet. Once an initial module or modules are delivered to the user, the application begins to execute while additional modules are streamed in the background. The modules are streamed to the user in an order which is selected to deliver the modules before they are required by the locally executing software. The sequence of streaming can be varied in
15 response to the manner in which the user operates the application to ensure that needed modules are delivered prior to use as often as possible.

In a newly developed application streaming methodology, described in co-pending U.S. Patent Application entitled "Method and System for Executing Network Streamed Applications", filed concurrently with the present application, the client system is provided
20 with client-side streaming support software which establishes a virtual file system ("VFS") and connects it to the client's operating system such that the virtual file system appears to be a storage device. The VFS is configured as a sparsely populated file system which appears to the operating system to contain the entire set of application files but, in reality, will typically

contain only portions of selected files. Client streaming functionality is provided to process streamlets or blocks of individual files and add them to the VFS as appropriate.

SUMMARY OF THE INVENTION

The present invention is directed to a method and system for preprocessing and
5 packaging application files for use by a network-based streaming application service provider
which is configured to stream the software to a client for client-side execution. The
application files are divided into sets of streamlets, each of which corresponds to a data block
in a particular application file at a particular offset and having a predefined length.
Preferably, the data blocks are equal in size to a code page size used during file reads by an
10 operating system expected to be present on a system executing the application. For
Microsoft Window's based systems, 4k code pages are used.

Each streamlet is then compressed and the compressed streamlets are packaged into a
streamlet repository. While the original boundaries between application files are not
preserved in this repository, index data is provided to permit specific streamlets to be
15 retrieved with reference to the source filename and an offset in that file. In addition, an
application file structure which indicates how the various files associated with an application
appear to a computer when the application is locally installed is determined and this
information packaged with the repository. With reference to the application file structure,
precompressed streamlets corresponding to specific code pages in a particular application file
20 can be selectively extracted from the streamlet repository.

According to a further aspect of the application preprocessing method, the application
is selectively installed on a suitable test machine. Prior to installation, a "snapshot" of the
environmental condition of the test machine, such as environmental variable settings,

contents of system control files, etc., is taken. This starting condition is compared with the environmental condition after the application has been installed and the environmental changes due to the application installation are determined. This information is recorded in an environmental install package. When the application is streamed to a client, the
5 environmental install package can be used to properly configure the client system during a virtually installation of the streaming application.

After the application has been installed on the test machine (or another machine), the application is started and the sequence in which the various file blocks are loaded are monitored during the application startup process. Those application streamlets which are
10 required to enable execution of the application to be initiated, and preferably those streamlets required to have the application run to a point where user interaction is required are identified. Those identified streamlets form a startup streamlet set which represents a minimal portion of the application which should be present on the client system for the application to begin execution.

15 According to a further aspect of the invention, in addition to packaging the application into a streamlet repository and identifying environmental install data and a startup streamlet set, the application is executed using test inputs and simulated or real user interaction and the sequence of code and data loads generated by an operating system as it executes the application are monitored. This information is used to generate a predictive
20 model of the order in which the application file blocks are loaded during execution. The predictive model can be used by a streaming application server to determine an optimal order in which to send streamlets to a client to minimize the likelihood that the application will

require a portion of an application file before the corresponding streamlet has been sent to the client.

Through use of the new methods, preprocessed application streaming packages can be easily and quickly generated for a variety of applications. The packages can include the streamlet repository, application file structure, environmental install package, startup set definition (or the startup streamlets themselves combined in a cluster), and predictive model. The packages can then easily be provided to one or more streaming application servers.

Advantageously, the streaming application package does not require any modifications to the application code or data itself. Further, because the application files are segmented into streamlets in a manner which is independent of the actual code or data content of the files, a wide variety of application packages can be easily prepared for streaming in this manner. In addition, the application streamlet repository as well as the additional elements of the application package can be formatted and stored in any manner suitable for retrieval on the server system. Thus, the server operating system and data storage environment can be selected without regard for whether it is compatible with the application's environment and the manner in which the application is stored. Finally, precompressing each streamlet prior to delivery to the server decreases the net size of the streamlet repository and increases the overall streaming rate without increasing server load by compression on-the-fly or the time required for the server to extract the streamlets from the repository.

The details of one or more embodiments of the invention are set forth in the accompanying drawings and the description below. Other features, objects, and advantages of the invention will be apparent from the description and drawings, and from the claims.

DESCRIPTION OF THE DRAWINGS

The foregoing and other features of the present invention will be more readily apparent from the following detailed description and drawings of illustrative embodiments of the invention in which:

Fig. 1 is a block diagram of an application streaming system;

Fig. 2 is a high-level flowchart of a method for packaging a set of application files into a streamlet repository;

Fig. 3 is an illustration of the segmentation of a sample application during processing in accordance with the invention;

Fig. 4 is a high-level flowchart of a method for determining the environmental changes made to a computing platform when an application is installed;

Fig. 5 is a high-level flowchart of a method for determining a startup streamlet set and generating a streaming prediction model for use in streaming the application; and

Fig. 6 is a sample usage graph representation of one form of a prediction model.

DETAILED DESCRIPTION OF THE INVENTION

Turning to Fig. 1, there is shown block diagram of a streaming application system 110, the performance of which can be improved through the use of preprocessed application packages produced in accordance with the present invention. The system includes a server 112 which can be accessed by one or more clients 114 via a data network 116, such as the Internet, an intranet, extranet or other TCP/IP based communication network, or other types of data networks, including wireless data networks.

The server 112 stores a streaming application package for use in streaming a software application to a client on request. As discussed in more detail below, to prepare an

application for streaming, the various files used for the application are divided into small segments called streamlets which are then stored, preferably in a compressed form, in a suitable library accessible to the server. In addition, a file structure specification for each hosted application which defines how the various files associated with an application appear to a computer when the application is locally installed is provided on the sever. An application's file structure specification can be packaged in a Startup Block which is sent to a client when streaming of the application begins. The Startup Block can also contain startup streamlet set which includes at least those streamlets containing the portions of the application required to enable execution of the application to be initiated, and preferably those streamlets required to begin application execution and have the application run to a point where user interaction is required. In addition, further application information, such as environmental variable settings, additions to system control files, and other system modifications or additions which may be required to "virtually install" the application can be provided.

Once the startup streamlet set is received and loaded at the client, the client initiates execution of the application. Concurrently, the server continues to push streamlets to the client in accordance with the predictive model. In addition, the server is responsive to fetch requests issued by clients to retrieve specific streamlets which have not yet been delivered. The streamlets can be forwarded to a client individually or grouped together and pushed in clusters as appropriate.

Various embodiments of the client system can be used. In one embodiment, the client system is configured with a virtual file system which appears to the client operating system as a local drive on which all the files needed by the streaming application reside. When a

required segment is not present on the client machine, the client issues a request for the appropriate streamlet(s) to the server. Usage information 120 can also be sent from the client 114 to the server 112 and can be used by the server to determine which streamlets to provide next.

5 Implementations may use a server system as disclosed in U.S. Patent Application entitled "Method And System For Streaming Software Applications to a Client" and filed concurrently with the present application. Implementations may also use a client system as disclosed in U.S. Patent Application entitled "Method and System for Executing Network Streamed Applications" and filed concurrently with the present application. The entire
10 contents of these application has been expressly incorporated by reference.

Fig. 2 shows a high-level flowchart of a method for packaging a set of application files into a streamlet repository and Fig. 3 is an illustration of the segmentation of a sample application during processing in accordance with the invention. With reference to Figs. 2 and 3, a software application that is installed on a computer system generally comprises one
15 or more files, such as executable files, DLLs, data files, system initialization files, etc., which are placed in one or more specified folders. Initially, the files which are used by the installed application are identified and the file structure of the application is determined. (Steps 200, 202).

A sample application FOOBAR is illustrated in Fig. 3 as it is installed on a computer
20 system. The application comprises a 100k executable file FOOBAR.EXE 300 and a 40k library file FOOBAR.DLL 502 , both of which are stored in application folder FOOBAR. In addition, a data file DEFAULT.DAT 304 is also provided and stored in a DATA subfolder. The file structure 306 can be represented in tree format as shown. Although not shown, the

application file structure can also include additional information about the files, such as a file size, version number or file date, write protect status, and other file-related meta-data.

Each of the application files are then divided into separate blocks. The division between blocks is made with reference to the manner in which the application file is likely to be read by the computer operating system when the client is executed. Preferably, the block sizes are each equal to a code page size used during file reads by an operating system expected to be present on a client system. For standard Microsoft Windows systems, files are generally read in four kilobytes blocks; the use of a block size equal to the operating system read size may be desirable. Since file system may load different types of files differently, it is possible for the block size to vary between different types of files and possibly even within different parts of a given file. However, in the implementation disclosed herein, the selected block size is constant within each file and across all files in the application. Fig. 3 shows the sample files divided into blocks 308, each of which is 4k in length. As will be appreciated, given a fixed block size, each block is uniquely associated with its source location with reference to the source file (e.g., a name and path) and an offset within that file.

After the application has been divided into blocks, each of the blocks is preferably compressed (step 206) and packaged into a streamlet repository along with a suitable access index (step 208). Various database or other random-access formats can be used for the repository such that each of the blocks can easily be individually extracted with reference to the index or in another manner. The streamlet repository is suitable for use in streaming the application from a server host to one or more clients, such as shown in Fig. 1.

This preprocessing permits the blocks from the application files to be stored in a file format or on a file system which differs from that the application is designed to run on. The streamlet repository can be used on a server which has a file system that is incompatible with the system on which the application will be executed. Particular streamlets can be extracted
5 from the repository by the server (or another system) without concern for the contents of the application files or the manner in which the files are structured.

Further, because the streamlets are in a compressed format, server resources do not need to be expended in compressing the streamlets prior to transmission to a client.

Although client resources are generally needed to decompress the streamlets, given the
10 comparatively slow nature of communication links, such as dial-up modems, ISDN and DSL services, and wireless connections, the benefits of compressed data transmissions outweigh the added processing costs.

The streamlet repository can be combined with at least the application file structure information to form a preprocessed application package. This package can be easily
15 distributed to streaming application servers and used by those servers to stream the application to suitable client users.

Some applications make changes to the overall computing environment when they are installed. Environmental changes include additions or modifications to registry file entries, control or initialization files (on computers using the Windows(tm) operating system, these
20 initialization files often have an ".ini" extension), file application mapping or bindings between different file types and associated applications, etc. In order for such an application to be streamed to a client and operate correctly, it may be necessary to make certain changes in the client's computing environment. Preferably, when such a streaming application is first

started, the server sends appropriate environmental information to the client along with suitable software (if necessary) for the client to the environmental information and modify the client system appropriately. This environmental modification information can be included in an environmental install package which is generated during preprocessing of the application and bundled with the application package for distribution.

Fig. 4 is a high-level flowchart of a method for determining the environmental changes made to a computing platform when an application is installed. Initially a client or model client system is prepared (step 400) and the initial state of the environmental settings is recorded (step 402). Next, the application is loaded onto the test system (step 404) and the final state of the environmental settings is recorded. (Step 406). The initial and final environmental states are then compared to identify the changes which were caused by installing the application. (Step 408) These changes are then recorded in a suitable format in one or more files which comprise the environmental install package for the application. For example, a single file or data object can be used within which, under different headings, are additions to the system registry, changes to specified control or ini files, etc. As will be appreciated, several different types of client systems may be used and respective environmental install packages generated for use in virtually installing a streaming application on that type of client system. The correct package to forward to a client can later be selected by the server.

According to yet a further aspect of the invention, and with reference to Fig. 5, the operation of the application itself, and particularly, the order in which the various blocks in the application files are loaded as the application is executed is monitored. (Steps 500-502). Various techniques to monitor or spy on operation system file load requests will be known to

those of skill in the art and any suitable technique can be used. Monitoring continues until the application has reached a designated startup point, such as a pause where the application awaits user input. (Step 504) The identified blocks themselves or their identities can then be compiled into an initial or startup block definition (step 506) which is included in the application package. When the server begins streaming the application to a client, the "InitBlock" contents can be forwarded to the client first to ensure that when the application is executed, sufficient portions of it have been provided for execution to begin without immediately indicating that additional data is required from the server. The number of blocks included in the InitBlock will, typically, be determined by balancing the desire to provide responsive application performance once the user begins interacting with the application (this criteria implies a greater number of blocks in the InitBlock) and the desire to minimize the time needed to initialize the application and to begin interaction with the user (this criteria implies a limited number of blocks so as to limit download delays).

In addition to monitoring the initial set of blocks loaded during individual application and during application suite execution, the application(s) can be executed on a suitable testbed and the overall sequence of block loads during execution monitored. (Steps 508-510). Various automated or manual testing scenarios can be performed and, preferably, all of the major application functions are executed. Monitoring may continue until the application has reached a designated startup point, such as where the application awaits user input (Step 504). Blocks may thereby be identified as belonging to the InitBlock Bundle of the application suite. When a client terminal connects to the server, the InitBlock Bundle can be the first content sent to the client to ensure that when application in the suite are executed,

The data load information is analyzed, perhaps in conjunction with information indicating the specific program state during various loads, to generate one or more predictive models (step 512). These models can then be included as part of the application package to be used by a server. When streaming the application, the server can use the model to
5 determine the order in which block or blocks are most likely to be loaded by the application when it is in a given state so that the streamlets most likely to be needed by the client executing the streaming application are sent to it first. The models are preferably provided in a format which can be modified by the server in accordance with the usage information and other information gathered during the streaming process itself.

10 Various statistical techniques can be used to analyze the sequence of loads generated by an operating system as it executes the application and generate a predictive model, and determine an optimal order to push the application streamlets to the client. In one embodiment, the predictive knowledge base can be viewed as a graph where a node is a user request or action (e.g. save, load) and an edge is the calculated probability that such a request
15 will be made. Behind each node are specifications of the application blocks which are loaded by the application when the request is made. A simple example of such a graph is shown in Fig. 6. By examining the links flowing from a given node, the system can easily determine the most likely future requests and so determine the streamlets which will be needed by the application to execute those requests.

20 It is common for software users to access a suite of related programs when performing a task. For example, to prepare a document, a user may need to use a word processing program, a spreadsheet program, and a drawing program. Software manufacturers may respond to this type of usage pattern by grouping applications into suites. For example,

the Microsoft Office(tm) suite of programs contains a grouping of applications for performing common office tasks.

Application service providers (ASPs), and other providers of software downloaded on-demand, may also wish to group applications into suites of related programs. To improve the usability of such application suites, it is desirable that the switching time when moving between applications be minimal. That is, when a user switches from a first to a second application in an ASP's suite, it is desirable that the user interface of the second application appears with a minimal delay (even if functionality of that application is initially restricted). The ability to rapidly initialize applications as the user switches between them may improve the user's perception of the ASP's and application's performance.

In an ASP environment, rapid switching between applications may be obtained by packaging startup blocks (i.e., InitBlocks) of multiple applications (or portions of the InitBlocks) into a package of interrelated InitBlocks (an "InitBlock Bundle"). The InitBlock Bundle contains software blocks that perform basic initialization task for the suite of applications. These initialization tasks can include the presentation of a basic user interface to the user (thus allowing the user to rapidly perceive activation of the application). Preferably, after the InitBlock Bundle is received by the client computer, it will be stored in relatively permanent memory (e.g., hard disk storage, EEPROM, or other relatively permanent storage) so that it can both be re-used in the future and easily updated when necessary. Among the advantages that can be achieved by using InitBlock Bundles are the ability to launch and switch between applications faster, improved user perception of service speed, the ability to subscribe to a portion of applications represented in an InitBlock Bundle, rapid access to the

other applications represented in the InitBlock upon subscription to those applications, improved usage profiling, and the ability to create new offerings based on user behavior.

After an InitBlock Bundle is received at a client terminal, a server may thereafter begin streaming of other blocks of the application suite. A predictive algorithm, such as the
5 algorithm disclosed in co-pending U.S. Patent Application Serial No. 09/120,575, entitled "Streaming Modules" and filed on July 22, 1998, may be used. When a user accesses an application represented in an InitBlock Bundle, usage data 120 may be sent to the server 112 indicating the application being used. In some implementations, the server may thereafter stream that application's blocks to ensure rapid response of the application being used.

10 A server can determine the blocks needed in an InitBlock Bundle by monitoring an application loading process (see Fig. 5), by monitoring streaming request, or by explicit designation from a software developer. In some cases, two or more applications may use a common set of initialization blocks. For example, two applications may use common code to initialize toolbars and menus. The initialization blocks can be composed of executable or
15 non-executable data, or a mix thereof. In some implementations, the InitBlock Bundle can include blocks for applications from different application service providers. This may be done where, for example, different application service providers have partnered to offer a set of services.

InitBlock Bundling may be determined, and/or service subscription offers made,
20 based on usage data collected as a user is executing applications at the client computer.

Usage data 120 can be sent to the server 112 indicating the applications and/or application features being accessed. This usage data may be stored and compared to a data in a statistical usage database 121 to determine application marketing opportunities. For example, based on

the usage data 120 and the statistical usage data 121, the system may determine that a particular application that the user does not subscribe to has features that are likely to be of interest to the user. In response, the system 112 may send data to the user's terminal causing a pop-up window or other display indication to appear. The display indication can contain marketing and subscription information offering the unsubscribed application to the user. The user may be able to click on the pop-up window to subscribe to the offered application (e.g., by clicking on an "Accept Offer" button or completing a subscription form). If the user decides to subscribe to the application, the server may respond by sending access control data to the client terminal enabling access to the unsubscribed application. For example, a decryption key may be sent.

Some example scenarios of InitBlock Bundle usage will now be described. In the following scenarios, three exemplary application service providers (ASP1, ASP2, and ASP3) are considered. ASP1 offers applications S1, S2, S3, S4 and has a user base consisting of subscribers U1, U2, U3. ASP2 offers applications S1, S2, S5, S6, S7 and has a user base consisting of subscribers U4, U5, U6, U7. ASP3 offers applications S2, S3, S6, S8, S9 and has a user base of U8, U9, U10, U11, and U12.

In scenario 1, Users U8, U9, U10 each subscribe to applications S2, S3, S6 offered by ASP3. Each of these users receives an InitBlock Bundle that includes the InitBlocks of applications S2, S3, and S6. In addition, the InitBlock Bundle sent to U8, U9, and U10 includes InitBlocks for other (currently unsubscribed) services S8 and S9 available from ASP3. User's U8, U9, and U10 will, therefore, be able to rapidly start any of applications S2, S3, S6. In addition, if the users later subscribe to S8 or S9, these services can be rapidly accessed using the InitBlock information previously sent in the InitBlock Bundle.

In scenario 2, users U8, U9, U10 each subscribe to applications S2, S3, S6 of ASP3.

Usage statistics data 121 indicates that 80% of users of applications S2 and S3 will respond

favorably to an offer for application S5. Application S5 is unavailable from ASP3, but may

be obtained from ASP2. To offer S5 to its subscribers, ASP3 may partner with ASP2 and

5 include the S5 InitBlock in the InitBlock Bundle sent from ASP3. Usage data 120 collected

at ASP3 may be used to account to ASP2 for usage of S5. In some cases, ASP3's server

system may provide only the S5 InitBlock to the user's client computer. Remaining blocks of

S5 may be obtained from a second server system under control of ASP2. To support this

division of applications across different server systems, the InitBlock may include additional

10 data pointing to the appropriate server system. Alternatively, control data may flow between

server systems operated by ASP2 and ASP3 to control and synchronize the streaming of

additional blocks of S5. A second alternative is for ASP3 and ASP2 to partner such that both

sets of applications are offered to all customers. Thus, an InitBlock Bundle offered by this

partnership including InitBlock data for applications S1, S2, S3, S4, S5, S6, and S7.

15 In scenario 3, a virtual aggregate ASP (ASP123) is created and monitors the usage of

all applications served by ASP1, ASP2, and ASP3. ASP123 monitors the usage data 120

generated by all of the ASPs and dynamically updates InitBlock Bundles using the sets of

InitBlock Bundles from the ASPs. ASP123 can access the usage database 121 and usage data

120 to form different InitBlock Bundles targeted to different users or groups. Thus, a

20 continuously customized collection of InitBlock Bundles may be created and distributed.

In cases in which multiple ASPs partner to offer services, usage data 120 can be

monitored to determine appropriate revenue accounting arrangements. In addition, the usage

data 120 can be used to offer new pricing models, such as dynamic pricing (i.e., time-

sensitive pricing) or per-feature pricing. Additionally, it can be used to create an integrated billing and software distribution system based on the behavior of users.

Applications developers commonly re-use program code and libraries among multiple applications. For example, application suites may share code to render a common look-and-feel of window, toolbar, and menu structures. As a result, different applications may embody common blocks of code. In many cases, this is true even for applications from different software vendors (for example, different vendors may each use the same library of code provided by the same third-party vendor). Download delays can be limited by providing a mechanism for sharing (i.e., reusing) common code blocks among different applications, including applications from different vendors. Advantageously, such a mechanism can operate even in the absence of an application developer's knowledge of this common code usage because the application itself knows what it needs. The ability to re-use common code blocks can help to minimize the size of InitBlocks and InitBlock bundles.

Reuse of streamed blocks can be enabled using a unique identifier exchanged between the streaming server and client. This unique identifier can be used to determine if the client already has a stored copy of a particular code block, or if the client needs the block of code to be sent. The client, upon receiving the unique identifier, can compare it to identifiers of already-received blocks. If the client determines that a needed block is already present at the client, the client can re-use the stored block (e.g., by copying it to a needed location). If the needed block is not present at the client, control data can be sent back to the server to cause the block to be streamed to the client.

A unique hash key may be used as an identifier of each of the blocks streamed from the server to the client. A block's hash key can be, e.g., a 32-bit, 64-bit, or 128-bit value

generated by processing each byte of a block using a hash algorithm. Many different hash algorithms can be used (e.g., digital signature algorithms, on-way hash algorithms, etc.). The hash algorithm and hash key size are preferably selected such that the hash key size is substantially smaller than the block size and, for any two different start blocks, two different hash keys will be produced. It is recognized that practical hash algorithms and key sizes will likely have some probability of producing the same hash key for different code blocks. If this occurs, execution errors may result; however, appropriate hash algorithm and key size selection can reduce this event to a statistically insignificant probability.

Fig. 7 illustrates a process that may be used by a streaming server to determine a set of code blocks needed to execute an application at the client computer. The process may begin with a message 701 from a server 112 containing unique identifiers (i.e., hash values H1-H3) of a set of code blocks. For example, the nine 4K blocks of the "foobar.dll" file (Fig. 3) may be uniquely identified by nine 64-bit hash values (one per 4K block). The message 701 may include all or a subset of these hash values; as shown message 701 includes three of the hash values (H1-H3). Upon receiving the message 701, the client computer may query a local database or table containing the hash values of all (or a subset of all) code blocks stored at the client. Based on this query, the client can determine the needed blocks and responds 702 by identifying which of the blocks H1-H3 are stored at the client (e.g., H1 and H3 in message 702). Conversely, the message 702 may indicate the needed blocks. In some cases, the client 702 may "find" blocks H1, H2 in unrelated code (i.e., code previously received for unrelated applications).

After receiving message 702, the streaming server processes the received data to determine a streaming order for those blocks identified in message 701 that are not located at

the client. The server may then begin the streaming of the needed block(s) 703 (i.e., block H2). In some implementations, additional blocks not identified in message 701 may also be streamed. After the block(s) have been received by the client 704, the server may again query the client 705. In some cases, the client can respond by indicating the desired streaming order of blocks. For example, the message 706 Indicates that the client already has a copy of the block identified by hash key H5, that the next blocks needed at the client is block H9 followed by H4. In response the server sends blocks H9 followed by H2 and then selects a streaming order for blocks H6-H8. In some implementations, the server may send control data to the client identifying a particular file by file-name or other conventional file identifier. The use of identification keys to determine needed blocks may be used for blocks in InitBlock Bundles, as well as for other blocks. Prior to sending an InitBlock bundle, the server can send key values to the client to determine whether particular blocks in the InitBlock Bundle are present at the client. If so, these blocks may be omitted from the InitBlock Bundle. When the InitBlock Bundle is transmitted to the client, data can be included in the InitBlock bundle (e.g., the key value and/or other control data) to indicate that the client terminal should reconstruct the full InitBlock Bundle by adding to it the already-stored blocks identified by the key values.

The invention may be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. Apparatus of the invention may be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor; and method steps of the invention may be performed by a programmable processor executing a program of instructions to perform functions of the invention by operating on input data and generating output. The

invention may advantageously be implemented in one or more computer programs that are executable on a programmable system including at least one programmable processor coupled to receive data and instructions from, and to transmit data and instructions to, a data storage system, at least one input device, and at least one output device. Each computer

5 program may be implemented in a high-level procedural or object-oriented programming language, or in assembly or machine language if desired; and in any case, the language may be a compiled or interpreted language. Suitable processors include, by way of example, both general and special purpose microprocessors. Generally, a processor will receive instructions and data from a read-only memory and/or a random access memory. Storage devices

10 suitable for tangibly embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, such as EPROM, EEPROM, and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM disks. Any of the foregoing may be supplemented by, or incorporated in, specially-designed ASICs (application-specific

15 integrated circuits).

A number of embodiments of the present invention have been described.

Nevertheless, it will be understood that various modifications may be made without departing from the spirit and scope of the invention. Accordingly, other embodiments are within the scope of the following claims.